UCRL-ID-120075

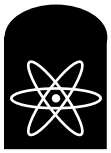
A Kernel Approach to Safety Systems

Prepared by

G. G. Preckshot

Prepared for

U.S. Nuclear Regulatory Commission



FESSP

Fission Energy and Systems Safety Program

Lawrence Livermore National Laboratory

This work was supported by the United States Nuclear Regulatory Commission under a Memorandum of Understanding with the United States Department of Energy, and performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

A Kernel Approach to Safety Systems

G. G. Preckshot

Manuscript Date: November 14, 1994

CONTENTS

A KERNEL APPROACH TO SAFETY SYSTEMS

1.0 INTRODUCTION

Software failures have become so much a subject of public concern that a popular general science publication (*Scientific American*) recently carried an article on the "software crisis" (Gibbs 1994). The litany of software failures is too long to repeat here, but respected members of the engineering community, in apparent contradiction of public wisdom, have stated "Indeed, a *simple* software-based system, in which the hardware is kept within its environmental constraints, and whose software is *simple* enough to have been subjected to a full validation and verification... can be expected to never fail" (Ward 1992) (italics added). The reason for the difference of opinion lies in the word *simple*.

1.1 Motivation

The motivation for this study is to find a way to specify software-based safety systems as simply as possible, but not simpler (Einstein). From a regulator's point of view, this is a worthwhile goal because the regulatory burden is eased and greater certainty of safety is achieved. From an equipment or system vendor's point of view, any approach that simplifies and speeds the regulatory approval process spends fewer vendor resources in administrative activities and allows more effort to be directed to product improvement and sales. The crux of the matter, however, is that, paradoxically, simple is extremely difficult to define.

1.2 Informal Definition

Informally, it is proposed to continue and specialize the concept of a "safety kernel" (Rushby 1989). Kernels have been around computer science since at least 1968 (Dijkstra). A safety kernel was proposed by Leveson in 1983, and an alternative safety kernel approach was described by Rushby in 1986. The choice of the word *kernel* is deliberate and is intended to convey informally the ideas of small, hard, and essential. A safety kernel is that part of the system, including software, that when all functions not essential to safety are taken away, remains. Because it is hard, it functions even when non-essential functions are disabled. This report will demonstrate how to define such a safety kernel unambiguously and formally. Practical aspects of real kernels will not be neglected. A glossary of mathematical logic symbols is included at the back of this report for those readers unfamiliar with this kind of symbology.

1.3 The Problems

There are two problems when attempting to specify any system "as simply as possible, but not simpler." *Not simpler* means that the specification is complete. *As simply as possible* means that nothing extra is included.

1.3.1 Completeness

Completeness is meaningless without a yardstick against which to judge. For safety systems, measures of completeness are typically taken from defined design basis events, failure modes and effects analyses, and mitigation responses required to respond to design basis events. Event sequences and fault trees used in probabilistic risk assessments provide a way to connect these three sets and to discover derivative requirements imposed by different combinations of failures and implementation idiosyncrasies.

1.3.2 Minimality

Minimality is the condition in which the functions of a safety system meet the requirements of completeness and legal impositions, but no more. A legal imposition, for example, is the requirement for diversity, which results in two ways of accomplishing the same end. A system can be minimal by construction, and provably minimal by backwards traceability analysis. In other words, every function of the safety system is traceable back to a primitive imposition or requirement derived from such impositions, and no function of the safety system is without antecedent.

1.4 Objectives

The objectives of this report are to produce a definition of a safety kernel that is compatible with regulatory practice, has practical application, and will be compatible with emerging formal methods for property proving and code generation.

1.4.1 Unambiguous Definition

A definition must be unambiguous so that it can be determined if a particular safety system is, or is not, a safety kernel. Furthermore, the definition should give rise to criteria for applicability of the approach, so that practitioners can tell whether the use of a safety kernel is appropriate to a particular safety problem, or it is not.

1.4.2 Compatibility with Regulatory Practice

Safety kernels, as defined herein, should be couched in terms of design bases, which is the current regulatory approach to safety in nuclear facilities. The design bases should be stated in terms of reactor parameter measurements and mitigation actuations, rather than accidents, since digital safety systems do not monitor accidents, but instead monitor parameters, and do not perform physical actions, but instead actuate equipment that does.

1.4.3 Practical Application

The safety kernel concept should have practical embodiments that take the results of risk analyses and physical modeling to a statement of the logic and decision sequencing that results in safety actions.

1.4.4 Compatibility with Emerging Formal Methods

The statement of logic and decision sequencing should be logically complete at the chosen abstraction level so that mechanical translation to a specific formal description is possible. It may be necessary to supply additional information to satisfy unanticipated requirements of the formal description, depending upon the purpose or intentions of the formal method chosen. However, properties of correctness, consistency, and sequencing should be invariant under the translation.

1.5 Report Organization

The history of kernels as a software construct is reviewed to discover if an unambiguous definition of "kernel" exists. Two approaches described by their authors as safety kernels are presented and one is selected for further development. A form is derived that is consistent with current nuclear industry practice and restrictions arising from the original kernel statement, and assumptions made during derivation are stated as criteria for applicability. Three areas of practical application are shown to meet the criteria. An approach using some PRA techniques is described for application of the safety kernel concept. Attributes of the method and those due to the design bases selected for a safety kernel are reviewed.

2.0 PREVIOUS WORK ON SOFTWARE KERNELS

Software kernels have been the focus of many workers in computer science, but consensus on a precise definition has been so elusive that some might almost consider kernels the Grail of operating system theory. Initial usage was in operating system development, and a noted paper by Dijkstra (1968) described a kernel as progressive shells of abstraction that hid the details of hardware from computer programmers. In the early 1970s, kernels were depicted as being the small, essential part of an operating system that had to be written in assembly (machine) language, and which performed services so basic to the functioning of all programs running on the machine, that they could be entrusted only to the kernel. A graphical illustration of this is shown in Figure 1. To support this notion, hardware designers included privileged machine states, accessible only to kernel software. At this writing, hardware support of kernel privilege may be the only constant in the operating system concept of kernel.

Also, from 1975 to 1990, kernels grew. The Unix "kernel" illustrates the great divergence in kernel definitions, with some versions exceeding 1,000 kilobytes in size (Vowler 1993). In comparison, Holt (1983) gives a Tunis kernel for the PDP-11 computer of less than 50 machine instructions (about 200 bytes). There is clearly a major difference in definition. The effect of this was not significant until operating system suppliers were confronted with the task of adapting

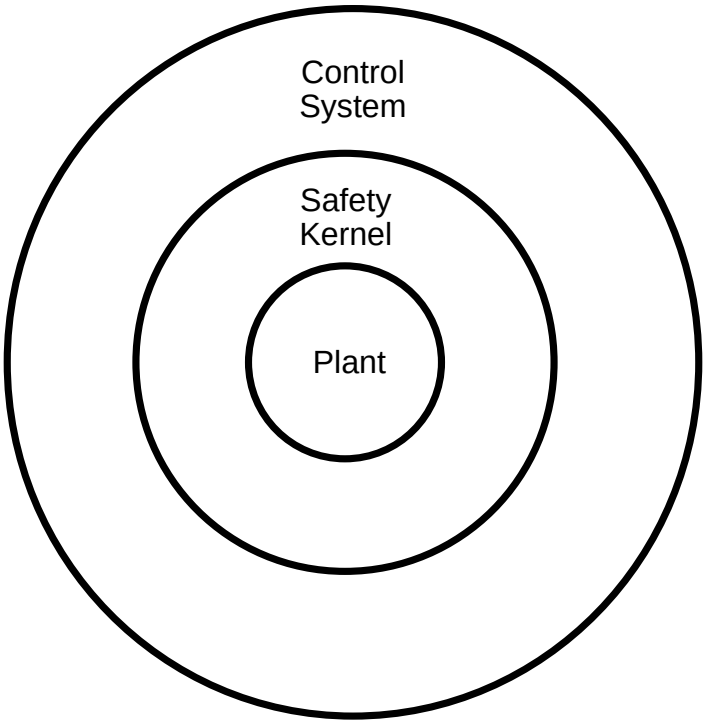


Figure 1. The kernel approach—kernels are "inside."

("porting") their operating systems to new computer systems. Confronted with major efforts both to adapt and to verify the adaptation of their operating system kernels to new computer systems, operating system suppliers have returned to earlier approaches to kernels, which they now call "micro kernels." The issues involved are very similar to those encountered in safety systems. For operating systems, these issues are separating and restricting the size and functions of the machine-dependent portion of an operating system so that adaptation to a new computer system and verification of the result can be accomplished with the use of minimal resources and time.

The first application of a kernel concept to safety systems is due to Leveson et al. (1983), where the kernel was envisioned primarily as a structuring concept, but critical concerns of independence and separation were not addressed. Rushby, coming from a background of information security, applied the concepts inherent in a "security kernel" to the problem of enforcing critical safety requirements (Rushby 1989). Rushby's approach both improved the rigor of kernel definition and directly addressed the issues of independence and separation (Rushby 1982).

In the following sections, the previously mentioned prior work is discussed in more detail, and the concepts pertinent to safety kernels are noted in italics.

2.1 T.H.E.

The T.H.E. operating system (Dijkstra 1968) used five layers of abstraction, often shown, as in Figure 1, as concentric layers or spherical shells, somewhat like the layers of an onion. The

innermost layer, surrounding the physical hardware, was termed the kernel, and the kernel provided services to the layer above, which contained abstractions, or idealizations, of real computer hardware to an easier-to-use logical model or "virtual machine." Although security and safety were not the primary objectives in the T.H.E. operating system, *the kernel restricted improper or unsafe use of computer system hardware by providing a limited set of functions that could be requested by non-kernel software*. This technique is effective so long as access to hardware can be limited to the kernel.

2.2 Tunis

Tunis (Holt 1983), a concurrent implementation of a Unix kernel, was typical of the minimal operating system kernel approach of the late 1970s, and has similarities to small, real-time kernels today. The approach of Dijkstra is continued in Tunis in the functions that the kernel implements: process synchronization , I/O control, and process control. *Strategic and policy modules, such as device management, memory management, the file system, operating system services, process scheduling, and utilities are not in the kernel*.

2.3 Unix

Unix is cited here as the epitome of large-kernel operating systems. Other commercial or proprietary systems have similarities, but these are not germane to tracing the development of software kernels. Versions of Unix produced under the aegis of the AT&T Corporation (Bach 1986) included strategic and policy modules within the kernel. Another source (Comer 1984) viewed the kernel as a module into which code could be collected so that it could be "isolated from user processes by the hardware's system call mechanism." Comer suggested configuration management as the means for controlling the multitude within the kernel. The only similarity between the large and minimal kernel approaches is that the software designated as "kernel" software executes in the privileged state provided by hardware designers so that operating system designers can control access to computer hardware. Recent "monolithic" Unix systems have kernels that are literally 1,000 times the size proposed for Tunis and like systems. *Monolithic Unix demonstrates by counterexample the error of entrusting too much to a kernel.*

2.4 Micro Kernels

Largely because of increasing difficulty in adapting monolithic-kernel operating systems to newer computer hardware, operating system suppliers have returned to earlier, more compact versions of kernels (Gien 1990). The two most popular approaches (1994) are the Mach kernel and the Chorus kernel. There are differences between the two, and between these two and other proposals, but process control and dispatch, interprocess communication and synchronization, interrupt handling, memory control mechanisms, and low-level network support are common to most micro kernel proposals. Other than these, functions or features are included based on performance or fiscal priorities. *We still lack a rigorous test for deciding what should be in the kernel and what should not*.

2.5 Safety Kernel

The objectives of early operating system kernels and current micro kernel approaches are attractive to safety system designers because of the limitation of concerns and the potential for separation and independence of the safety system from other systems. These qualities are important because they limit fault propagation, improve safety system reliability, and make calculation of risks easier. There have been two attempts to apply the kernel approach to safety.

2.5.1 Leveson et al. (1983)

Leveson et al. propose a kernel as a structuring and modularization technique embodied as "a set of mechanisms to provide support for the detection of and recovery from safety-critical errors, and a set of policies to govern the application software's use of these mechanisms." Detection of errors is by the execution of "assertions" (essentially first-order predicates) embedded in non-kernel software. Recovery (or mitigatory action) occurs by kernel-mediated execution of other, non-kernel, recovery software. While the benefits of modularization are well known, *the safety attributes of the system do not depend exclusively on the safety kernel, nor are the limits of validity for application of the Leveson et al. safety kernel established.*

2.5.2 Rushby (1989)

Rushby proposes a kernel consistent with the original Dijkstra vision, in that through the abstractions (functions) it provides, a negative assertion about safety ("bad things don't happen") is maintained by limitations upon what non-kernel software (or systems) can do. A yardstick for minimality is formed by the logical statement of the safety attributes that the kernel maintains, regardless of the actions of other systems:

$$\forall a \in op; P(a)$$

where *op* is the set of all sequences of invocations of functions provided by the kernel, and $P(a)$ is a predicate over the input/output behavior of *op* (i.e., safety response with respect to a). Rushby gives conditions for validity of the proposed kernel approach. *There is a formal statement of safety attributes, $P(a)$, the safety attributes depend exclusively upon the kernel, and there are tests for applicability of the approach.*

3.0 SAFETY KERNEL DEFINITION

A safety kernel definition should satisfy the mathematical requirement of rigor, yet be accessible to the practitioner. To do this, the definition will be accompanied by informal discussion of why each step is taken and why it is necessary. The first such topic is "why is a definition necessary at all?" There are several reasons, the first being that practitioners should have tests available before the fact to decide whether the safety kernel approach is applicable to their safety problem. A good definition will give rise to criteria for applicability. A second reason is that practitioners and regulators should be able to recognize when deviations are taken from the safety kernel approach, so that results from the theory are not inappropriately applied. Third,

the definition should be couched in terms and techniques familiar to at least some practitioners so that there is a path from the definition to industry practice. The approach to the definition will be described, followed by the definition, thence followed by developed criteria for application.

3.1 Approach

The starting point for the definition is the second order predicate expression taken from Rushby (1989), repeated here for convenience:

$$\forall a \in op; P(a)$$

This is applicable if the following two requirements are satisfied:

1. The properties of interest at the system level must be present at the kernel level, and,
2. Those properties must be expressed by a second-order assertion of the (above) form (or, as a special case, in the form of an invariant on the system state).

In the development given by Rushby for security kernels, the security kernel enforces $P(a)$ through the functions it provides to non-kernel software. To extend this to safety kernels, it is necessary to recognize that sequences of safety parameter measurement sets play the same role as sequences of function invocations do for security kernel theory, as long as a requirement for dynamical fidelity (stated below) is met. This does not exclude the possibility that a safety kernel may provide restricted functions in a manner similar to security kernels. For safety systems, requirement one states that parameters important to safety or functions that should be restricted for safety purposes must be available to, or provided by, the safety kernel. Requirement two states that a model based on sequences of parameters must have sufficient fidelity to the safety problem that the physical safety goals are met. Another requirement, which is implied by the predicate expression, is that the safety kernel must have absolute priority to take safety actions in spite of what other systems may do. As a matter of history, this priority has been conferred on operating system kernels through hardware support. The condition is satisfied in a similar manner by many extant safety systems. For example, a reactor protection system has absolute priority for reactor shutdown because its sensors are independent and it has access to preemptive reactivity control mechanisms.

Reactor safety systems can satisfy the general requirements for the safety kernel approach, but the application of $P(a)$ is obscure. The goal of definition is therefore to expand in terms meaningful to practitioners in the art of reactor safety. Specifically, $P(a)$ should be stated in terms of reactor safety parameters, design bases, and safety responses (or mitigations).

3.2 Definition

A second-order safety kernel ensures that a second-order logical safety assertion $P(a^k)$ is always true:

$$\forall a^k \in op; P(a^k)$$

where op is the set of sequences of sets of safety parameter measurements,

a^k is a sequence of sets of measurements available at interval k,, where a_i^k denotes a sequence of measurements by instrument i.

$P(a^k)$ is a predicate (logical expression) that is true when the plant is in the safe operating regime or a design basis event has occurred and the required mitigation functions have been initiated.

If b^k are safety parameter measurement set sequences of length one, the history of the safety system can be made explicit in the familiar form of the state machine. If the state machine is finite, its next state depends at most upon p past values of b . New state, s^{k+1}, is described by a "next state" function F

$$\forall b^k \in op, \forall s^k \in S;$$
$$s^{k+1} = F(b^k, s^k) \ .$$

This is a realistic description of a digital system, which perforce is a finite state machine. The predicate, $P(a^k)$, can then be written in terms of $b, s,$ and F, extending over all sequences of sets of safety parameter measurements of length less than or equal to *p+1*,

$$\forall b^k, b^{k-1} \in op, \forall s^{k-1} \in S; P(b^k, F(b^{k-1}, s^{k-1}))$$

where op is the set of sequences of sets of safety parameter measurements,

b^k, b^{k-1} are the *k*th measurement and the previous measurement set,

S is the domain of safety kernel state,

$s^{k-1}$is the previous state, including as many as *p-1* previous parameter measurements $b^i, k-p \le i \le k-2$,

$P(b^k, F(b^{k-1}, s^{k-1}))$ is a predicate (logical expression) that is true when the plant is in the safe operating regime or a design basis event has occurred and the required mitigation functions have been initiated.

Note that time is not an explicit variable and a^k $(b^k, b^{k-1}, b^{k-2}, \ldots, b^{k-p})$ is a sequence of safety parameter measurements of at most length *p+1*. $P(b^k, F(b^{k-1}, s^{k-1}))$ is therefore stationary. This is a restriction on models described by the expression; safety qualities are stationary and depend at most upon *p+1* measurements. This means that the plant safety transfer function (the function describing how the plant reacts to mitigation efforts) must be a limited function of time. In other words, the plant safety response is adequately described by the current state of

the safety system (which may include *p* previous parameter measurements) and the current plant parameter measurement set. An illustrative counter-example is given by the Chernobyl accident. At Chernobyl, because of unknown internal states, the sign of reactivity insertion of the control rods was effectively an unknown function of time. An action taken to make the reactor safe—rod insertion—had the opposite effect.

There is an additional restriction related to the fidelity of the model; the sequence of measurements $b^k, b^{k-1}, b^{k-2}, \ldots, b^{k-p}$ adequately models the dynamics of the physical system so that if $P(b^k, F(b^{k-1}, s^{k-1}))$ is true, radioactive release will not occur.

To further specialize $P(b^k, F(b^{k-1}, s^{k-1}))$ to reactor safety practice, the idea of a mitigation selector, $M(b^k, s^k)$, is introduced. M selects mitigation functions appropriate to values of b^k and s^k when b^k and s^k diverge into a design basis set. Otherwise, M selects no mitigation functions. More precisely, M is a relation from $op \times S$ to $\mathbf{M}$, the mitigation means available to the safety kernel for actuation—in other words, the safety kernel's outputs. A condition for irreversibility of safety kernel action, which satisfies the IEEE Std 603 requirement for completion of protection system action, can be stated for safety parameter measurement set sequences of length $\leq p+1$

$$b^k, b^n \in op; s^k, s^n \in S;$$
$$M(b^n, s^n) \subseteq M(b^k, s^k) \text{ for } k \geq n$$

which is a condition upon $F(b^k, s^k)$ because

$$M(b^k, s^k) = M(b^k, F(b^{k-1}, s^{k-1}))$$

$P(b^k, F(b^{k-1}, s^{k-1}))$ can thus be rewritten in terms of $b, s, M,$ and F

$$P(b^k, F(b^{k-1}, s^{k-1}), M(b^k, F(b^{k-1}, s^{k-1})))$$

which is a useful form of $P(a^k)$ for practitioners. $b, s, M,$ and F describe the logical requirements for a safety kernel under the assumption of a set of design bases, *D*, and $P(a^k)$ indicates the attributes of formal methods that will be useful either to describe a model of the system, or to demonstrate correctness.

The set of design bases, *D*, can be represented as points in a subset of $O \times \mathbf{M}$

$$d^k = \{b^k, s^k, m^k\} \in D \subseteq O \times M;$$
$$' \; m^k = M(b^k, s^k) \cup \{\Delta\}.$$

A particular design basis, D_a, represents a subset of D associated with a particular subset of instruments, $\{a_q^k, a_r^k, \ldots, a_v^k\} = I_a^k$, when

$$\forall\, d^j, d^k \in D_a$$
$$\text{if}\quad M(b^j, s^j) = M(b^k, s^k).$$

This is compatible with the definition of a design basis given in 10 CFR 50.2, which defines design bases as "that information which identifies the specific functions to be performed by a structure, system, or component of a facility, and the specific values or ranges of values chosen for controlling parameters as reference bounds for design. These values may be (1) restraints derived from generally accepted 'state of the art' practices for achieving functional goals, or (2) requirements derived from analysis (based on calculation and/or experiments) of the effects of a postulated accident for which a structure, system, or component must meet its functional goals." Current NRC practice is that the design bases for reactor protection systems consist of parameter ranges (b^k), operating states (s^k), and associated functions (m^k) that are agreed upon between NRC's Systems and Safety Analysis Division (DSSA) and vendor counterparts.

3.3 Criteria for Applicability

The criteria for applicability of the kernel approach are:

Observability:

Parameters are observable by the safety kernel, which, when controlled within bounds set by *stationarity* and *fidelity*, will enforce physical safety goals.

Stationarity:

The safety response is at most a function of limited history.

Fidelity:

At most *p+1* equally spaced observations are required to determine that physical safety goals are enforced.

Priority:

The safety kernel has absolute priority, enforced by hardware.

Effectiveness:

The mitigation means available for actuation by the safety kernel are eventually effective in controlling *observable* parameters within bounds set by *stationarity* and *fidelity*. Eventually means within the dynamic bounds set by physical safety goals.

Irreversibility:

A class of safety kernels is irreversible if, absent intervention,

$$M(b^n, s^n) \tilde{O} M(b^k, s^k) \text{ for } k \geq n$$

Design Basis:

The design basis for a safety kernel is a set, stated as observables, system states, and functions.

Coverage:

A safety kernel is provable only with regard to the design bases used to develop the second-order logical predicate $P(a)$. However, intersection between the design basis set and observables for other events not in the design basis will result in robustness to failures within the safety kernel and coverage outside the provable set. Diversity both of observables and response functions increases the likelihood that non-design-basis events will be detected and adequately mitigated.

4.0 SAFETY KERNEL APPLICABILITY

The safety kernel approach, although severely restricted for some applications, fits well with many hazardous systems with well-defined design bases, and stationary mitigation responses. It may also be appropriate for subsets of more difficult problems, where some design bases can be defined that fit the safety kernel criteria described in a previous section. In such cases, solving known safety problems with the kernel approach may leave resources to attack more difficult parts that have greater uncertainties. Three areas of application immediately suggest themselves.

4.1 Reactor Safety Systems

U.S. reactor safety systems, which this work directly addresses, satisfy the listed criteria, and are required to be irreversible with small exceptions (reset and automatic depressurization are two). There is regulator interest in reducing or limiting the complexity of safety-related systems both because focused application of regulator resources results in more assurance for the same expenditures and because the regulators do not become involved in side issues.

4.2 Medical Systems

Some medical systems, among which are those that deliver treatment to patients, can satisfy safety-kernel criteria. A notable bad example is the Therac-25 (Leveson & ....), which would have benefited from the discipline required to incorporate an independent, preemptive safety kernel. However, considering the possible managerial and institutional failures involved in this series of accidents, any broad claims that a safety kernel would have prevented the Therac-25

incidents is speculative[1]. The Federal Food and Drug Agency is currently investigating ways of preventing future Therac-25s, and safety kernels may be a viable way to give this agency greater control over, and greater assurance about, the safety of a subset of medical equipment and instrumentation.

4.3 Vehicle Systems

Automated highway systems are now under development. Not much is certain at this time except that the task is so great, and American commercial interests so strong, that any system that results will be fielded by many contractors and used by many shadetree mechanics. Safety will be an issue. Safety kernels offer a way to enforce at least some limits, and tie these limits directly to risk studies.

5.0 PRACTICAL APPLICATION

The predicate

$$P(b^k, F(b^{k-1}, s^{k-1}), M(b^k, F(b^{k-1}, s^{k-1}))$$

is a daunting expression, and a reasonable question is "how is it practically applied?" That question will be addressed in a sequel to this report, in which techniques now used in probabilistic risk assessment (PRA) will be applied to bridging the gap between input events and output actuations so that the predicate P remains true. Real-time digital systems perform short sequences of processing steps in response to input events. PRA techniques model sequential or dependent events as event trees, with non-sequential logic (where sequence of execution is irrelevant) being modeled in fault trees. These techniques appear to be applicable to safety kernels, particularly because of the close connection between risk assessment and design basis events.

6.0 KERNEL ATTRIBUTES

Attributes of the safety kernel are reviewed here from the total system perspective, and from the perspective of a subsystem that is a safety kernel, as required by the statement of work for this task.

---

[1]The Therac-25 failed so often in operation that it is unlikely that operators would have tolerated repeated safety system function. There was an institutional failure in that the maker did not recognize and repair deficiencies. Safety systems are only effective if the exercise of their function is relatively rare, and notable because of this rarity.

6.1 System

A system may consist of a number of safety kernels, or divisions, which, taken in combination, satisfy the criteria to be a safety kernel.

6.1.1 Hardware Support

Hardware supports priority, independence, observability, and effectiveness. The system may be configured as a collection of safety kernels, each of which has absolute priority to actuate safety features, or as a set of divisions, some subset of which (for example two out of three or two out of four) has absolute priority. In the latter case, the priority criterion is met by the system as a whole, but not by subsystems (divisions).

6.1.2 Priority

No other system can intervene to prevent the overall safety kernel system from performing its functions.

6.1.3 Independence

Because of the priority requirement, at the system level a safety kernel is independent of other reactor systems. Because of the observability and effectiveness requirements, a safety kernel is independent of the effects of failures or events defined in the design bases.

6.1.4 Stationarity

The safe direction for responses available to a safety kernel is a function of limited time history. For example, inserting control rods into a nuclear reactor should always move the reactor toward a more-safe condition, regardless of the previous time history of the reactor. Violation of this criterion invalidates the safety kernel approach as restricted in this work.

6.1.5 Irreversibility

A safety kernel is an irreversible safety kernel, or has an irreversible subset, if mitigation, once actuated, cannot be reversed by the safety kernel.

6.1.6 Single Failures

Proof against single failures is part of the design bases from which the predicate, P, is derived. For example, the single-failure criterion can be met by redundant safety kernels, each of which has absolute priority to actuate safety functions. However, this approach usually has unacceptable impact on operational availability. Alternatively, the single-failure criterion can be met by redundant divisions, any two of which have absolute priority to actuate safety functions. In this case, no single division has priority to actuate safety functions, and the single-failure criterion is expressed in P as logical combinations of individual division actuation decisions, and is therefore explicit in P.

6.1.7 Minimum Functions

Safety kernel functions are minimal with respect to the design bases, and the objective of elaboration by event (or decision) trees and fault (or success) trees is to ensure that the logic remains minimal as constraints of the implementation technology are imposed.

6.1.8 Diversity

Diversity resides in the selection of design bases from which the predicate, P, is derived. The use of PRA techniques as a design tool to elaborate the logic in P enables quantitative assessment of diversity by assigning correlated failures to non-diverse portions of the elaborated logic or events. This is a significant reason for taking the PRA approach.

6.1.9 Reliability

The reliability contribution of the safety kernel approach comes about because of greater simplicity and more precise software requirements. Other reliability-enhancing techniques are part of the design bases. For instance, redundancy in the form of multiple, independent safety kernels, or a division structure. Diversity provides reliability through overlapping parameters or mitigation means. Real accidents or incidents usually produce excursions of several parameters, and can be mitigated by one of several means. By choosing the design bases so that several mitigation means are actuated, either simultaneously or coordinated in time, the probability that the safety kernel response will be effective is increased. Defense-in-depth, where it involves equipment or software under control or part of the safety kernel, is also part of the safety kernel design basis. However, when other, non-safety, systems are permitted to compensate rare common-mode failures in the safety kernel as a defense-in-depth, this is not part of the safety kernel design basis.

6.1.10 Status Indication

From a human-factors standpoint, adequate status indication is a safety requirement to permit human beings to function as diverse backup to automatic safety systems, and also to reduce the probability of inappropriate human actions. However, status indication is separable into two constituents: status observation (by display hardware or subsystems), and status display. Status display is normally done by independent hardware, or independent subsystems which are not subject to the same failures as may disable safety actuation systems. From the point of view of a safety kernel, it is only necessary to show that connected status display hardware, or independent status display subsystems, do not prevent the safety kernel from maintaining $P(b^k, F(b^{k-1}, s^{k-1}), M(b^k, F(b^{k-1}, s^{k-1}))$ true.

System-level status indication differs from subsystem-level in that system-level actuations are displayed which may be the result of logical combinations of subsystem actions. The display system itself may be designed using safety kernel methodology.

6.2 Subsystem

A subsystem may be a safety kernel itself if it fulfills all of the safety kernel criteria. However, a subsystem may fulfill safety kernel criteria only in combination with other subsystems, as in the case of protection system divisions that require agreement of at least two divisions to actuate mitigation.

6.2.1 Independence

Independence of subsystems from non-safety systems is required by the priority criterion if, in combination, they constitute a safety kernel or individually they are safety kernels. Independence of subsystems from each other is required by the priority criterion if the subsystems are individually safety kernels, and by the priority and effectiveness criteria if, in combination, several subsystems constitute a safety kernel. For instance, if a vote of two out of four subsystems (divisions) preemptively causes a safety actuation, no other division can nullify the vote of two. Therefore, divisions must be individually independent.

6.2.2 Fail-to-Safe

Fail-to-safe or fail-as-is is set by the design bases for the safety kernel.

6.2.3 Minimum Functions

Subsystem functions are minimal with respect to the design bases, and the objective of elaboration by event (or decision) trees and fault (or success) trees is to ensure that the logic remains minimal as constraints of the implementation technology are imposed.

6.2.4 Reliability

The reliability contribution of safety kernel subsystems comes about because of greater simplicity and more precise software requirements, which enable the use of formal methods or enhance the effect of systematic software engineering practices. Other, system-level, reliability-enhancing techniques such as redundancy and defense-in-depth do not reside in subsystems, but in the architecture of subsystem interconnection. Diversity, part of the design bases, is effective at the subsystem level.

6.2.5 Status Indication

As mentioned above under systems, status indication is separable into two constituents, status observation (by display hardware or subsystems), and status display. The status display may itself be an independent subsystem.

6.2.6 Proofs of Correctness

Proving correctness starts with demonstrating that criteria for applicability are satisfied. Then, for input sequences of length p + 1 or less, $P(b^k, F(b^{k-1}, s^{k-1}), M(b^k, F(b^{k-1}, s^{k-1}))$ is shown to hold for outputs.

6.2.7 Timing

For each input to the output that maintains $P(b^k, F(b^{k-1}, s^{k-1}), M(b^k, F(b^{k-1}, s^{k-1}))$ true, the elapsed time remains within the physical safety dynamics of the design basis.

7.0 CONCLUSIONS

The history of kernels as a software construct was reviewed. Kernels were found to have a varied history and still to lack unambiguous definition. An approach by Rushby for the design of security kernels was selected as a starting point, and modified and developed to suit a restricted set of safety applications. A form was derived that is consistent with current nuclear industry practice. The restrictions were stated as criteria for applicability, and three areas of practical application were shown to meet the criteria. An approach using some PRA techniques was described for application. The attributes due to the safety kernel approach and those due to the design base selection were noted.

GLOSSARY OF MATHEMATICAL SYMBOLS

Theorem Symbols

"	For any
$	There exists
'	Such that
fi	Implies
\	Therefore

Logic Symbols

$a < b$	a less than b
$a £ b$	a less than or equal to b
$a \geq b$	a greater than or equal to b
$a > b$	a greater than b
ÿ a	not a
a ⁄ b	a or b
a Ÿ b	a and b
$a \approx b$	a exclusive - or b

Set Symbols

W, Y	*Sets*
W Õ Y	W *is a subset of* Y
W Ã Y	W *is a proper subset of* Y
o Œ W	*o is a member of* W
p œ W	*p is not a member of* W
W » Y	W *union* Y
W « Y	W *intersection* Y
W ¥ Y	(*o*, *p*) Œ W ¥ Y fi *o* Œ W, *p* Œ Y

REFERENCES

Maurice J. Bach, *The Design of the Unix Operating System*, Prentice-Hall, 1986.

Douglas Comer, *Operating System Design: The XINU Approach*, Prentice-Hall, 1984.

E. W. Dijkstra, "The Structure of the T.H.E. Multiprogramming System," *CACM*, 11, 5 (May 1968), pp. 341–346.

W. Wayt Gibbs, "Software's Chronic Crisis," *Scientific American*, September 1994, pp. 86–95.

Michel Gien, "Micro-Kernel Design," *Unix Review* V8 No. 11, November 1990, p. 58.

D. Harel et al., "Statemate: A working environment for the development of complex reactive systems," I*EEE Trans. Software Engineering*, 16 (4), April 1990.

R. C. Holt, *Concurrent Euclid, the Unix™ System, and Tunis*, Addison-Wesley Publishing Company, Inc., 1983.

R. C. Holt, E. D. Lazowska, G. S. Graham, M. A. Scott, *Structured Concurrent Programming With Operating Systems Applications*, Addison-Wesley Publishing Co., 1978.

IRR (Institute for Risk Research), "Generic Problem Competition," International Symposium on Design and Review of Software Controlled Safety-Related Systems, University of Waterloo, Waterloo, Canada, Revision 1, 20 January 1992.

Matthew S. Jaffe and Nancy G. Leveson, "Completeness, Robustness, and Safety in Real-Time Software Requirements Specification," 11th International Conference on Software Engineering, Pittsburgh, PA, May 1989, pp. 302-311.

Nancy G. Leveson et al., "Requirements Specification for Process-Control Systems," University of California, Irvine, Technical Report 92-106, November 10, 1992.

N. G. Leveson,....., "Design for Safe Software," *Proc. AIAA 21st Aerospace Sciences Meeting*, Reno, Nevada, January 1983.

J. Rushby, "Kernels for Safety?," in *Safe & Secure Computing Systems*, T. Anderson Ed., Blackwell Scientific Publications, 1989, pp. 210–220.

John Rushby, "Formal Methods and Digital Systems Validation for Airborne Systems," NASA Contractor Report 4551, December 1993.

John Rushby, "Proof of Separability—a Verification Technique for a Class of Security Kernels," *Proc. 5th International Symposium on Programming*, Turin Italy, April 1982, in *Lecture Notes in Computer Science* Volume 137, Springer-Verlag, pp. 352–367.

K. D. Russell et al., Systems Analysis Programs for Hands-On Integrated Reliability Evaluations (SAPHIRE) Version 5.0, NUREG/CR-6116, Vols 1–8, December 1993.

Ivan Selin, "ACRS Member Warns of Safety Threats From NRC Handling of Digital I&C," *Inside N.R.C.*, August 23, 1993.

Julia Vowler, "....." *Computer Weekly*, October 21, 1993, p. 48.

David A. Ward, "Digital Instrumentation and Control System Reliability," Memorandum from David A. Ward, Chairman, Advisory Committee on Reactor Safeguards to Ivan Selin, Chairman, U. S. Nuclear Regulatory Commission, September 16, 1992.